Politechnika Świętokrzyska
Wydział Elektrotechniki, Automatyki i Informatyki

Zespół:
Kot Jarosław
Prusicki Jakub

Łamacz haseł

Projekt zespołowy
na studiach stacjonarnych
o kierunku **Informatyka**
Stopień **II**

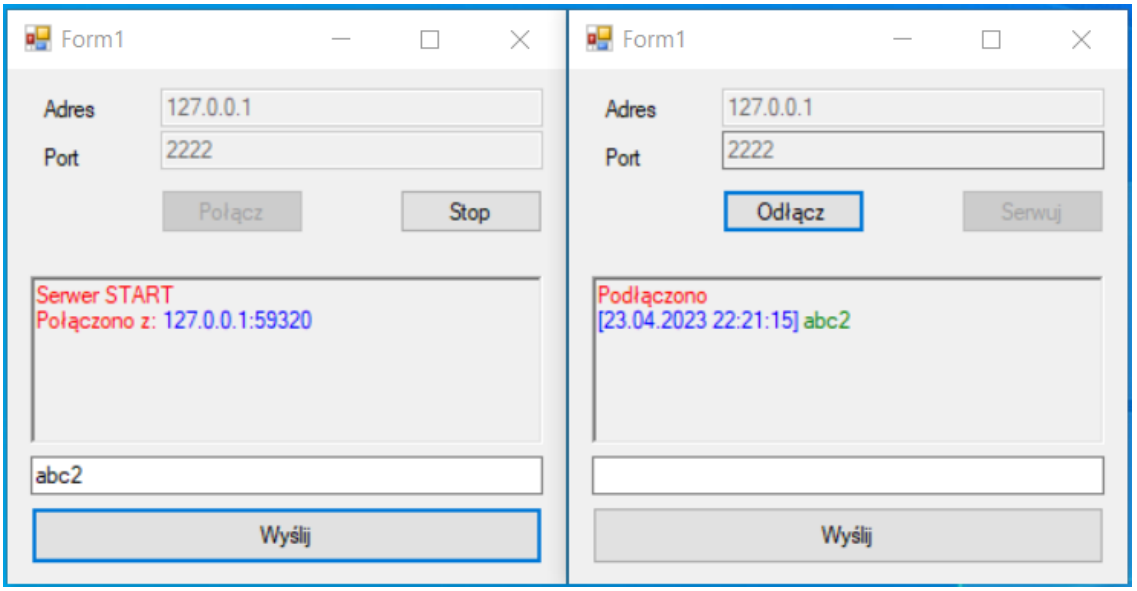
Opiekun projektu:
dr inż. Paweł Paduch

Kielce, 2023

SPIS TREŚCI

1. Architektura i technologie.

Do wykonania projektu zdecydowano się wykorzystać kod źródłowy udostępnionego już rozwiązania na platformie *achilles.tu.kielce.pl* przez opiekuna projektu. Jest to rozproszony komunikator tekstowy wykorzystujący połączeniowy protokół TCP. Zaletą tego konkretnego podejścia jest uniwersalność zaprojektowanego rozwiązania - odpowiednie przyciski *serwuj* oraz *połącz* pozwalają na opóźnienie decyzji, który z programów ma być serwerem. GUI aplikacji zostało wykonane z użyciem formatek okienkowych WinForms, będących z resztą częścią .Net Framework.



Rys. 1.1 Okienka aplikacji.

Podstawowymi elementami projektu są: struktura *Komunikat* oraz klasa *Klient*. Pozwalają one na prawidłowe przesyłanie treści w prezentowanym powyżej komunikatorze. Podczas wykonywania prac nad rozpraszaniem to właśnie ich modyfikacje będą miały największe znaczenie.

```
[Serializable]
public struct Komunikat
{
    public string tresc;
    public bool wazna;
    public string nadawca;
    public DateTime czasNadania;
    public DateTime czasOdbioru;
}
```

Rys. 1.2. Struktura komunikat.

```
class Klient
{
    public Thread watek;
    public TcpClient tcpKlient;
}
```

Rys. 1.3. Klasa Klient.

2. Algorytm Brute Force.

2.1. Implementacja.

```
namespace WindowsFormsApplication1
{
    class BruteForce
    {
        private static String password; //521ab
        public static StringBuilder str = new StringBuilder("");
        private static int min = 32, max = 127;
        private static DateTime start;

        public BruteForce(String password1)
        {
            password = password1;
        }

        public void RunAlghoritm()
        {
            start = DateTime.Now;

            while (true)
            {
                str.Append((char)min);

                for (int i = 0; i < str.Length - 1; i++)
                {
                    for (int j = min; j < max; j++)
                    {
                        str[i] = (char)j;
                        bruteForceAlghoritm(i + 1);
                    }
                }
            }
        }
    public void bruteForceAlghoritm(int index)
    {
        for (int i = min; i < max; i++)
        {
            str[index] = (char)i;

            if (index < str.Length - 1)
            {
                bruteForceAlghoritm(index + 1);
            }

            //Console.WriteLine(str);

            if (str.ToString().Equals(password))
            {
                DateTime stop = DateTime.Now;
                TimeSpan czasWykonania = stop - start;
                int czasLiczbowy = Convert.ToInt32(czasWykonania.TotalMilliseconds);
                StringBuilder str1 = new StringBuilder("");
                str1.Append("" + str);
                str1.Append(";" + czasLiczbowy + "ms");
                SaveResultToFile(str1 + "");
            }
        }
    }
```

Rys. 2.1 Algorytm Brute Force.

Cały algorytm mieści się w jednej klasie **BruteForce** i jest uruchamiany za pomocą metody ***RunAlghoritm()***. Program jest początkową wersją implementacji, która będzie rozbudowywana w celu poprawy jej obsługi (konfiguracja parametrów) co może mieć znaczący wpływ na zmianę wydajności działania samego algorytmu. Najważniejszymi atrybutami tej klasy są pola: **password, min** oraz **max**. Pierwszy z nich służy oczywiście do obsługi samego hasła, natomiast kolejne wyznaczają odpowiednio początek i koniec alfabetu. Są to liczby, ponieważ w tym programie odpowiadają one znakom z tablicy ASCII. Najczęściej używane zostały przedstawione w tabeli poniżej.

DEC	HEX	Znak	DEC	HEX	Znak	DEC	HEX	Znak	DEC	HEX	Znak
0	00	Null	32	20	Spacja	64	40	@	96	60	`
1	01	Start Of Heading	33	21	!	65	41	A	97	61	a
2	02	Start of Text	34	22	"	66	42	B	98	62	b
3	03	End of Text	35	23	#	67	43	C	99	63	c
4	04	End of Transmission	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal Tab	41	29	)	73	49	I	105	69	i
10	0A	Line Feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical Tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form Feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage Return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift Out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift In	47	2F	/	79	4F	O	111	6F	o
16	10	Data Link Escape	48	30	0	80	50	P	112	70	p
17	11	Device Control 1 (XON)	49	31	1	81	51	Q	113	71	q
18	12	Device Control 2	50	32	2	82	52	R	114	72	r
19	13	Device Control 3 (XOFF)	51	33	3	83	53	S	115	73	s
20	14	Device Control 4	52	34	4	84	54	T	116	74	t
21	15	Negative Acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous Idle	54	36	6	86	56	V	118	76	v
23	17	End of Transmission Block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of Medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File Separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group Separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record Separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit Separator	63	3F	?	95	5F	_	127	7F	Delete

Rys. 2.2 Tablica znaków ASCII.

Ponadto możemy dostrzec mierzenie czasu, które rozpoczyna się tuż po wywołaniu samej metody, a kończy w ostatniej instrukcji warunkowej - stwierdzającej czy dane hasło zostało złamane. Czas wykonania zapisywany jest w **milisekundach** w postaci liczbowej. Następnie hasło wraz z wartością czasu operacji jest zapisywane do pliku (możliwe, że format tekstowy będzie podlegał jeszcze modyfikacjom, bądź całkowitym zamianie np. na format typu excel). Sam plik tworzony jest w sposób uniwersalny na pulpicie wykonywanego sprzętu komputerowego. Co pozwala uelastycznić rozwiązanie i zaoszczędzić czasu na zmiany samych ścieżek, a skupić się na poważniejszych kwestiach obsługi samego programu. Oczywiście jeżeli taki plik już istnieje jest on już tylko nadpisywany. Kod zapisu wyników programu zaprezentowano na poniższym listingu.

```
public void SaveResultToFile(string text)
{
    string path1 = Environment.GetFolderPath(Environment.SpecialFolder.Desktop).ToString() + "\\passwords" + ".txt";

    DateTime thisMoment = DateTime.Now;
    if (!File.Exists(path1))
    {
        using (StreamWriter sw = File.CreateText(path1))
        {
            sw.WriteLine(text);
        }
    }
    else
    {
        using (StreamWriter sw = File.AppendText(path1))
        {
            sw.WriteLine(text);
        }
    }
}
```

Rys. 2.3 Zapis haseł do pliku.

2.2. Wady i zalety.

Zalety:

- łatwość zrozumienia i zaimplementowania
- uniwersalność - algorytm może być stosowany do rozwiązywania różnego rodzaju problemów, w tym problemów wielowymiarowych
- nie wymaga dodatkowych założeń
- możliwość stosowania nawet w przypadku problemów nieliniowych

Wady:

- mniejsza dokładność w porównaniu do innych metod numerycznych
- wymaga dużego nakładu obliczeniowego
- wymaga wyznaczenia dokładnej funkcji sił, która może być trudna do znalezienia w niektórych przypadkach

2.3. Inne rozwiązania.

Aby wybrać odpowiednią wersję algorytmu, sprawdzono dotychczasowe rozwiązania opublikowane już w zasobach internetu. Najważniejsze z nich, które pomogły w zbudowaniu algorytmu pochodzą z następujących źródeł:

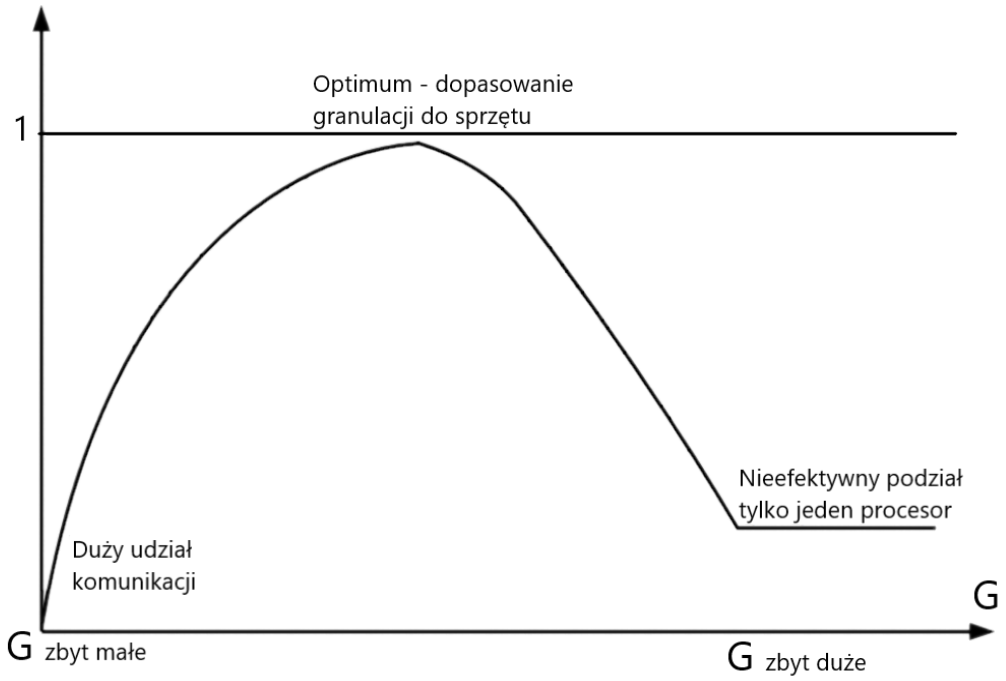
- Dot Net Office[1]
- Użytkownik serwisu github.com *jwoschitz*[2]
- C # Corner[3]

Oczywistym jest fakt, że nasz kod będzie nadal modyfikowany, będą implementowane dodatkowe funkcje, a te istniejące rozwijane, jednak na ten moment efekty prostego i poprawnie działającego algorytmu wystarczyły aby przetestować program przesyłający pojedyncze hasło od klienta do serwera aby ten mógł je złamać i zapisać do opisywanego już wcześniej pliku.

3. Wydajność

Ziarnistość projektu to poziom podziału pracy w projekcie, który określa, na jakie mniejsze części jest podzielony projekt i jakie zależności występują między nimi. W kontekście programowania współbieżnego, ziarnistość projektu odnosi się do sposobu podziału pracy pomiędzy wątki lub procesy. Wydajność ziarnistości projektu zależy przede wszystkim od rodzaju zadania oraz architektury systemu, w którym jest realizowane. W ogólnym przypadku, mniejsza ziarnistość (czyli mniejsze podziały pracy) może przyspieszyć wykonanie zadania, ale może też wprowadzić koszty synchronizacji i komunikacji między wątkami lub procesami. Z kolei większa ziarnistość (większe podziały pracy) może zmniejszyć koszty synchronizacji i komunikacji, ale może też wprowadzić koszty narzutu, związanego z tworzeniem i zarządzaniem dodatkowymi wątkami lub procesami. W przypadku projektów o dużym obciążeniu procesora, mniejsza ziarnistość jest zwykle bardziej wydajna, ponieważ zapewnia lepsze wykorzystanie

zasobów sprzętowych. W przypadku projektów o dużym obciążeniu sieciowym lub wejścia-wyjścia, większa ziarnistość może być bardziej wydajna, ponieważ zmniejsza ilość operacji synchronizacji i komunikacji. Ostatecznie, wybór odpowiedniej ziarnistości projektu zależy od indywidualnych wymagań projektu oraz dostępnych zasobów sprzętowych. Warto pamiętać, że optymalna ziarnistość projektu może być osiągnięta tylko poprzez testowanie i analizę wyników działania systemu w różnych konfiguracjach.



Rys. 2.4.1 Wykres doboru ziarnistości[4].

W ogólnym przypadku ziarnistość kodu jest pojęciem względnym, ściśle związanym z określona architekturą systemu wieloprocesorowego. Kod wykonany na systemie z pamięcią dzieloną sklasyfikowany jako gruboziarnisty, może zostać uznany za drobnoziarnisty w systemie z pamięci rozproszoną. Zgodnie z tym, co zostało przedstawione, w zależności od architektury koszt zarządzania wielozadaniowością (np. zarządzanie wątkami) może znacznie różnić się od siebie. W zależności od typu użytej transformacji pętli moliwe jest uzyskanie podanej ziarnistości kodu dla wybranej architektury. Tabela 2.4.2 przedstawia podział transformacji pętli programowych w zależności od możliwej do uzyskania ziarnistości.

Transformacja	Ziarnistość[1]	
	drobno-	grubo-
Zmiana kolejności wykonani pętli (ang. *loop interchange*)	●	●
Rozszerzenie skalaru (ang. *scalar expansion*)	●	○
Zmiana nazw zmienny skalarnych (ang. *scalar renaming*)	●	○
Zmiana nazw zmiennych tablicowych (ang. *array renaming*)	●	○
Podział węzłów (ang. *node splitting*)	●	○
Redukcja (ang. *reduction*)	●	○
Podział zmiennych indeksowych (ang. *index-set splitting*)	●	○
Przekoszenie pętli (ang. *loop skewing*)	●	●
Prywatyzacja (ang. *privatization*)	○	●
Podział pętli (ang. *loop distribution*)	○	●
Wyrównanie (ang. *alignment*)	○	●
Replikacja kodu (ang. *code replication*)	○	●
Łączenie pętli (ang. *loop fusion*)	○	●
Odwrócenie wykonania pętli (ang. *loop reversal*)	○	●
Wielowymiarowe łączenie pętli (ang. *multilevel loop fusion*)	○	●

Tabela 2.4.2 Tabela transformacji do uzyskania określonej ziarnistości[5]

LITERATURA

[1] Kod algorytmu brute force

https://www.dotnetoffice.com/2022/10/brute-force-algorithm-in-c.html

[2] Kod algorytmu brute force - 2.

https://gist.github.com/jwoschitz/1129249

[3] Kod algorytmu brute force - 3.

https://www.c-sharpcorner.com/article/how-using-brutal-force-could-improve-code-quality

[4] Wykres doboru ziarnistości.

https://achilles.tu.kielce.pl/portal/Members/332bd2d158a24964b2b98a7fc2879842/programowanie-wspolbiezne/wyklady/1-wstep/wyk1_wprowadzenie.pdf

[5] Tabela transformacji do uzyskania określonej ziarnistości.

http://zbc.ksiaznica.szczecin.pl/Content/4621/PHD_Krzysztof_Siedlecki.pdf